依赖注入技术

依赖注入， DI（ dependency injection ），是一系列用于开发解耦合代码的软件设计原则和模式。依赖注入是设计模式范畴的技术,主要用于降低应用程序代码的耦合度。解耦合代码是依赖注入的关键。依赖注入是其中一种解耦合方法。依赖注入实现的三种方式：构造函数注入（Contructor Injection）、setter注入、接口注入。依赖注入降低了依赖和被依赖类型间的耦合，在修改被依赖的类型实现时，不需要修改依赖类型的实现，同时，对于依赖类型的测试，可以更方便的使用mocking object替代原有的被依赖类型，以达到对依赖对象独立进行单元测试的目的。

设计模式

设计模式是软件开发人员在软件开发过程中面临的一般问题的解决方案，是一套被反复使用的、多数人知晓的、经过分类编目的、代码设计经验的总结。使用设计模式是为了重用代码、让代码更容易被他人理解、保证代码可靠性。根据设计模式的参考书 Design Patterns - Elements of Reusable Object-Oriented Software 中所提到的，总共有 23 种设计模式。这些模式可以分为三大类：创建型模式（Creational Patterns）、结构型模式（Structural Patterns）、行为型模式（Behavioral Patterns）。

|  |  |  |
| --- | --- | --- |
| 创建型模式：提供了一种在创建对象的同时隐藏创建逻辑的方式，而不是使用 new 运算符直接实例化对象。这使得程序在判断针对某个给定实例需要创建哪些对象时更加灵活。 | 结构型模式：关注类和对象的组合。继承的概念被用来组合接口和定义组合对象获得新功能的方式 | 行为型模式：特别关注对象之间的通信。 |
| 工厂模式（Factory Pattern） | 适配器模式（Adapter Pattern） | 责任链模式（Chain of Responsibility Pattern） |
| 抽象工厂模式（Abstract Factory Pattern） | 桥接模式（Bridge Pattern） | 命令模式（Command Pattern） |
| 单例模式（Singleton Pattern） | 过滤器模式（Filter、Criteria Pattern） | 解释器模式（Interpreter Pattern） |
| 建造者模式（Builder Pattern） | 组合模式（Composite Pattern） | 迭代器模式（Iterator Pattern） |
| 原型模式（Prototype Pattern） | 装饰器模式（Decorator Pattern） | 中介者模式（Mediator Pattern） |
|  | 外观模式（Facade Pattern） | 备忘录模式（Memento Pattern） |
|  | 享元模式（Flyweight Pattern） | 观察者模式（Observer Pattern） |
|  | 代理模式（Proxy Pattern） | 状态模式（State Pattern） |
|  |  | 空对象模式（Null Object Pattern） |
|  |  | 策略模式（Strategy Pattern） |
|  |  | 模板模式（Template Pattern） |
|  |  | 访问者模式（Visitor Pattern） |

原型模式（Prototype Pattern）是用于创建重复的对象，同时又能保证性能。这种类型的设计模式属于创建型模式，它提供了一种创建对象的最佳方式。这种模式是实现了一个原型接口，该接口用于创建当前对象的克隆。当直接创建对象的代价比较大时，则采用这种模式。例如，一个对象需要在一个高代价的数据库操作之后被创建。我们可以缓存该对象，在下一个请求时返回它的克隆，在需要的时候更新数据库，以此来减少数据库调用。实例： 1、细胞分裂。 2、JAVA 中的 Object clone() 方法。 优点： 性能提高。 逃避构造函数的约束。 缺点： 配备克隆方法需要对类的功能进行通盘考虑，这对于全新的类不是很难，但对于已有的类不一定很容易，特别当一个类引用不支持串行化的间接对象，或者引用含有循环结构的时候。 必须实现 Cloneable 接口。

组合模式（Composite Pattern），又叫部分整体模式，是用于把一组相似的对象当作一个单一的对象。组合模式依据树形结构来组合对象，用来表示部分以及整体层次。这种类型的设计模式属于结构型模式，它创建了对象组的树形结构。这种模式创建了一个包含自己对象组的类。该类提供了修改相同对象组的方式。应用实例： 1、算术表达式包括操作数、操作符和另一个操作数，其中，另一个操作符也可以是操作数、操作符和另一个操作数。 2、在 JAVA AWT 和 SWING 中，对于 Button 和 Checkbox 是树叶，Container 是树枝。 优点： 高层模块调用简单。 节点自由增加。 缺点：在使用组合模式时，其叶子和树枝的声明都是实现类，而不是接口，违反了依赖倒置原则。

享元模式（Flyweight Pattern）主要用于减少创建对象的数量，以减少内存占用和提高性能。这种类型的设计模式属于结构型模式，它提供了减少对象数量从而改善应用所需的对象结构的方式。享元模式尝试重用现有的同类对象，如果未找到匹配的对象，则创建新对象。应用实例： 1、JAVA 中的 String，如果有则返回，如果没有则创建一个字符串保存在字符串缓存池里面。 2、数据库的数据池。 优点：大大减少对象的创建，降低系统的内存，使效率提高。缺点：提高了系统的复杂度，需要分离出外部状态和内部状态，而且外部状态具有固有化的性质，不应该随着内部状态的变化而变化，否则会造成系统的混乱。

责任链模式（Chain of Responsibility Pattern）为请求创建了一个接收者对象的链。这种模式给予请求的类型，对请求的发送者和接收者进行解耦。这种类型的设计模式属于行为型模式。在这种模式中，通常每个接收者都包含对另一个接收者的引用。如果一个对象不能处理该请求，那么它会把相同的请求传给下一个接收者，依此类推。应用实例：1、JS 中的事件冒泡。 2、JAVA WEB 中 Apache Tomcat 对 Encoding 的处理，Struts2 的拦截器，jsp servlet 的 Filter。 优点： 降低耦合度。它将请求的发送者和接收者解耦。 简化了对象。使得对象不需要知道链的结构。增强给对象指派职责的灵活性。通过改变链内的成员或者调动它们的次序，允许动态地新增或者删除责任。 增加新的请求处理类很方便。 缺点：不能保证请求一定被接收。 系统性能将受到一定影响，而且在进行代码调试时不太方便，可能会造成循环调用。 可能不容易观察运行时的特征，有碍于除错。